VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Bára Groňová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Bára Groňová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ComProMiS, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek**

Konzultant bakalářské práce: Ing. Roman Potoczný

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

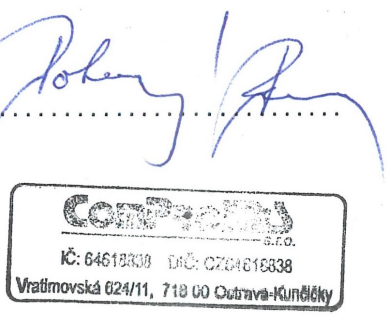
prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 24. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 24. dubna 2019 .....



Ráda bych tímto poděkovala Ing. Romanovi Potocznému za příležitost a odborné vedení, bez kterého by tato práce nemohla vzniknout, a celému kolektivu firmy ComProMis, s.r.o. za pomoc při řešení problémů. Poděkování patří také mému vedoucímu práce Ing. Pavlu Dohnálkovi, Ph.D. za pomoc při jejím vypracování.

Abstrakt

Cílem této práce je shrnout průběh odborné praxe ve firmě ComProMiS, s.r.o. V úvodu uvádím odborné zaměření firmy a popis pracovní pozice. Hlavní náplní praxe byla úprava a rozšíření plánovací aplikace RePlan, proto se tato práce zabývá právě tímto úkolem. Popisuji původní aplikaci, její nedostatky a požadavky zadavatele na výsledný systém. Následně se zabývám implementací těchto požadavků. Zaměřila jsem se především na odůvodnění zvoleného postupu vzhledem ke koncovým uživatelům s přihlédnutím na náročnost aplikace. Uvádím také popis použitých algoritmů a problémy, které se v průběhu vývoje vyskytly. V závěrečné části uvádím využité dovednosti a znalosti, které jsem nabyla v průběhu studia, ale také ty, které mi v průběhu praxe chyběly. Na úplný závěr práce zařazuji celkové zhodnocení odborné praxe.

Klíčová slova: odborná praxe, replan, compromis, plánování

Abstract

The aim of this work is to summarize course of professional practice in company ComProMiS, s.r.o. In the introduction I mention professional focus of the company and description of job position. Main focus of the practice was modification and extension of planning application RePlan, which is why this thesis deals with this task. I describe original application, its shortcomings and the client's requirements for the final system. Next I deal with implementation of these requirements. I focused mainly on the reasoning of the chosen approach with respect to the end users, taking into account difficulty of the application. I also describe the algorithms used and the problems that have occurred during the development. In the final part I mention used skills and knowledge that I gained during my study, but also those that I missed during my practice. At the end of the thesis I include an overall evaluation of professional practice.

Key Words: professional practice, replan, compromis, planning

Obsah

Seznam použitých zkratek a symbolů

CSS	–	Cascade Style Sheet
HTML	–	Hyper Text Markup Language
UI	–	User Interface
ASP	–	Active Server Pages
ORM	–	Objektově-relační mapper
EF	–	Entity Framework
JS	–	Javascript
SQL	–	Standard Query Language
MS	–	Microsoft
UI	–	User Interface
API	–	Application Program Interface
JSON	–	JavaScript Object Notation
AJAX	–	Asynchronous Javascript And XML
MVC	–	Model-View-Controller
DOM	–	Document Object Model
PJ2	–	Programovací jazyky 2
DAIS	–	Databázové a informační systémy
UDBS	–	Úvod do databázových systémů
VIS	–	Vývoj informačních systémů
SPJA	–	Skriptovací programovací jazyky a jejich aplikace
URO	–	Uživatelská rozhraní
IE	–	Internet Explorer
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V rámci své bakalářské práce jsem se rozhodla absolvovat odbornou praxi ve firmě převážně kvůli nabytí zkušeností mimo akademickou půdu. Ze seznamu nabízených praxí mne zaujala firma ComProMiS, s.r.o. sídlící v Ostravě. Práce se měla týkat vývoje webové aplikace v .NET MVC s využitím EF nad databází MS SQL. V rámci výuky mě vývoj webových aplikací zaujal a s databázemi jsem měla rovněž kladné zkušenosti. Již při pohovoru byl nastíněn přibližný obsah práce a byla jsem přijata. Dostala jsem za úkol upravit a dokončit aplikaci Replan, kterou v minulosti zpracoval v rámci bakalářské praxe jiný student. Tato práce se tedy bude týkat převážně této aplikace, od analýzy nedostatků a problémů, přes jejich řešení až ke zhodnocení celého průběhu praxe a shrnutí využitých a chybějících znalostí. Zaměřuji se v ní především na popis a odůvodnění použitých postupů. Při řešení jednotlivých úkolů jsem se soustředila jak na funkčnost a jednoduchost z pohledu koncového uživatele, tak na nenáročnost výsledné aplikace.

1.1 Popis odborného zaměření firmy

Firma ComProMiS, s.r.o. se zaměřuje na zakázkovou tvorbu software s následnou údržbou a je partnerem společnosti Software AG. Jako jedni z mála v České Republice a na Slovensku umí tvořit software na operačním systému Unix a na bázi databáze Adabas a vývojového prostředí Natural. V současné době se soustředí převážně na vývoj v prostředí Windows na bázi .NET Framework, jako jsou ASP.NET, MVC či C# a v databázových prostředích MS SQL Server nebo Oracle. V neposlední řadě se zaměřují také na mobilní aplikace. Většina vyvíjeného softwaru cílí do podpory nákupu, zpracování zakázek, výměny informací mezi centrálami a pobočkami či podpory servisních aktivit. Mezi jejich nejvýznamnější zákazníky patří mezinárodní společnost OTIS.

1.2 Popis pracovní pozice

Ve firmě jsem byla zaměstnaná na pozici programátor/analytik. Má práce byla převážně samostatná s občasnou pomocí ochotných kolegů, kterou jsem párkrát vyhledala při dlouhodobějším neúspěšném řešení některých problémů. Hlavní náplní mé práce bylo dokončení plánovací aplikace RePlan pro interní účely firmy. Mým vedoucím byl Ing. Roman Potoczný, který byl současně zadavatelem požadavků na výslednou aplikaci. Pracovala jsem s technologiemi ASP.NET, MVC, EF, MS SQL, JS, HTML, CSS a dalšími.

2 Analýza stávajícího systému a jeho nedostatků

V prvních dnech praxe jsem se seznamovala s aplikací Replan, její funkcionalitou a implementací. Jednalo se o na první pohled funkční, ale nepoužívanou aplikaci pro plánování projektů zaměstnancům. Uživatel si mohl založit vlastní firmu, v ní následně centra a projekty. Mohl také pozvat či registrovat nové zaměstnance a těm poté plánovat práci na projektech. Nedílnou součástí bylo nastavení pracovní doby zaměstnanců, bez které nebylo možné plánovat. Aplikace byla řešena pomocí třívrstvé architektury v jazyce C#, kdy prezentační vrstvu tvořil projekt typu ASP.NET MVC a doménovou a datovou vrstvu projekty typu Class library. Využívala EF 6 s Code First přístupem a databázový systém MySQL.

Prvním zjevným nedostatkem bylo nepřehledné zpracování uživatelského rozhraní, které uživatele nenavádělo k jednoduchému a intuitivnímu používání. Dalším a závažnějším problémem bylo špatné plánování na časové ose. Implementované algoritmy pro rozpočítávání hodin překrývajících se projektů vykazovaly vysokou chybovost a zobrazení na časové ose za pomoci vizualizační knihovny vis.js bylo nepřehledné. Bylo tedy zjevné, že hlavní funkcionalita, pro kterou byla tato aplikace vyvíjena je nevyhovující. Na mnoha místech v aplikaci byla načítána z databáze nepotřebná data. Jednotlivé funkce si je často nepředávaly, ale každá je získávala sama. To způsobovalo časté přístupy do databáze, které nebyly nutné, což značně zpomalovalo celou aplikaci.

2.1 Požadavky na výsledný systém

Zadavatelem požadavků na aplikaci byl můj konzultant. Hned v úvodu byly zadány základní požadavky ohledně zamýšleného využití a technologií. Výsledný systém měl být nasazen na firemní server využívající MS SQL Server. Měl být primárně vyvíjen pro interní používání ve firmě s možným budoucím rozšířením pro využívání veřejností. Využívat se měl především pro přidělování úkolů zaměstnancům a přehled vedení o nakládání s časem. Dále pro účely pravidelných porad, na kterých se probírá aktuální a očekávaný stav zakázek (v aplikaci jako projekty), kdo na nich bude pracovat a v jakém rozsahu.

Po úvodní konzultaci ohledně požadavků na vyvíjený systém jsem se začala zabývat úpravou UI a návaznosti aplikace. V rámci této práce jsem si důkladně osvojila implementaci všech částí a zároveň odhalovala další problémy a získávala svou představu o výsledném systému. Poté byly požadavky a očekávání zadavatele prodiskutovány důkladněji spolu s mými návrhy a poznatky. Hlavním požadavkem tedy bylo přehledné a jednoduché plánování. Plány měly být zadávány jako interval dní s požadovaným počtem hodin na jeden den. Měla být implementována možnost zadání více úkolů na stejný den, kdy aplikace rovnoměrně přidělí plánům pracovní dobu zaměstnance. Časové osy měly být k dispozici celkem tři. Jedna pro zaměstnance implementovaná tak, aby jednoznačně viděl na čem má pracovat a v jakém rozsahu. Dále pro celou firmu, kde budou zobrazeny naplánované projekty jednotlivým zaměstnancům. Poslední měla být časová osa centra zobrazující, kdy na jednotlivých projektech pracuje který zaměstnanec. Pro účely

porad byl požadavek na tisk časových os a přehledu projektů. Měla být také implementována možnost pozvat k registraci nového uživatele přes email.

2.2 Hlavní úpravy vyplývající z analýzy a požadavků

Z důkladné diskuze o požadavcích zadavatele a mých poznatcích a návrzích vyplynuly následující nejdůležitější body, na kterých jsem měla nadále pracovat. Jejich řešení a porovnání s původní implementací shrnu v následující kapitole.

1. Změna z MySQL na SQL Server
2. Úprava uživatelského rozhraní
3. Možnost tisku časových os
4. Rozšíření možností práce s projekty
5. Přidání možnosti pozvat k registraci a do firmy neregistrované uživatele
6. Kompletně nové plánování projektů zaměstnancům

Zároveň bylo dohodnuto, že kostra aplikace, zabezpečení a většina použitých technologií je pro zadaný účel vyhovující.

3 Provedené změny a rozšíření systému

V této kapitole budou popsány hlavní změny a rozšíření původního systému, které byly nastíněny v analýze. U každé uvádím porovnání s předchozím řešením, důvod změny a implementaci včetně problémů, které se při vývoji dané části naskytly. Kapitola je rozdělena na změny na jednotlivých vrstvách a v závěru pojednává o přidaných komponentách.

3.1 Datová vrstva

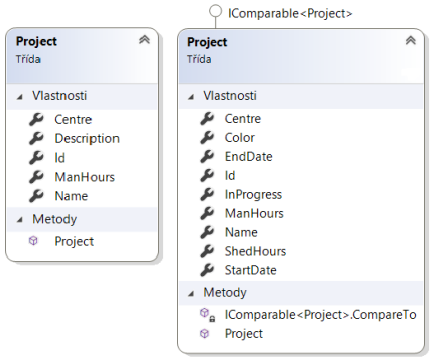
Změny na této vrstvě se týkaly především úprav struktury databáze. Většina byla patrná a implementovaná hned po analýze. Aplikace využívá EF s Code First přístupem, takže změny byly aplikovány při prvním vytvoření databáze. Některé úpravy však byly prováděny v průběhu vývoje. Abych zachovala má testovací data využila jsem nástroj Migrace obsažený v EF [5].

3.1.1 Změna z MySQL na SQL Server

Aplikace měla být nasazena na firemním serveru, který využívá MS SQL Server. Z tohoto důvodu byla nutná změna databázového systému. Aplikace nepoužívala žádné uložené procedury ani triggery a tudíž díky použití EF byla tato změna velmi jednoduchá pouze změnou v konfiguračním souboru a doinstalováním příslušného balíčku. Aplikace nebyla v minulosti používána, tudíž nebyla potřeba migrace žádných dat. V průběhu vývoje byl využíván lokální server na mém počítači a při nasazování aplikace byla databáze i se základními daty pomocí zálohy přenesena na firemní server.

3.1.2 Úprava tabulky Project

Databázová tabulka uchovávající projekty byla rozšířena o pět sloupců. Pro přehlednější zobrazování plánů na časové ose byla přidána informace o barvě projektu, kvůli usnadnění plánování a správy projektů byly přidány sloupce obsahující počet již naplánovaných hodin, počáteční a koncové datum a také byla tabulka rozšířena o sloupec indikující, zda je již projekt ukončen.



Obrázek 1: Tabulka projektů před a po změně

Počet naplánovaných hodin a data jsou využívány pouze informativně. Není tedy v aplikaci nijak omezeno plánování přesahující očekávané hodiny, nebo interval mezi začátkem a koncem projektu. Informace, zda je již projekt dokončen, se zdála jako nezbytná. Bez ní nebylo možné filtrování a vyřazení z plánování. Jedinou možností bylo celý projekt smazat, což ovšem smaže z databáze i všechny naplánované části a z hlediska manažera projektu by takto přicházel o zpětný pohled na výkony zaměstnanců a nakládání s časem. V databázi je tato hodnota vyjádřena jako boolean (bit) kdy výchozí hodnota je true (1) a představuje, že projekt probíhá. Ukončit projekt je možné při jeho editaci a tato změna je kdykoli stejným způsobem vratná. Způsobí, že projekt se již nebude nabízet v seznamu při plánování a ve výpisu všech projektů bude řazen na konci nehledě na počet naplánovaných hodin.

Třída projekt v aplikaci implementuje rozhraní IComparable. Metoda CompareTo zařizuje seřazení projektů na základě procent nenaplánovaných hodin a ukončené projekty řadí na konec.

```
int IComparable<Project>.CompareTo(Project other)
    {
        if (!other.InProgress && InProgress) return -1;
        if (other.InProgress && !InProgress) return 1;

        if ((ShedHours / ManHours) < (other.ShedHours / other.ManHours))
            return -1;
        if ((ShedHours / ManHours) > (other.ShedHours / other.ManHours))
            return 1;
        return 0;
    }
```

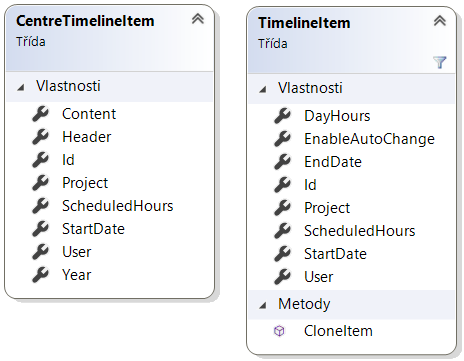
Výpis 1: Implementace rozhraní IComparable třídy Projekt

3.1.3 Změna způsobu ukládání plánů

V původní aplikaci byly plány ukládány ve formě v jaké je zadal uživatel. Obsahovaly počáteční datum a počet požadovaných hodin. Počet dní a k nim přiřazených hodin byl počítán až při vykreslování. Tento způsob sice znamenal nejnižší množství dat v databázi, ale navzájem se ovlivňující plány musely být děleny při každém novém vykreslení časové osy, což způsobovalo značné zpomalení systému a také nekonzistentnost při zobrazení plánování pro firmu, uživatele či oddělení, protože se vždy počítalo pouze s plány, které byly zrovna zobrazovány.

V nové aplikaci neměly být plány zadávány s finálním počtem hodin, ale v intervalu od kdy do kdy má zaměstnanec na daném projektu pracovat a požadovaný počet hodin denně, které se následně upravily v závislosti na nastavení pracovní doby a ostatních plánech. Ukládání přesně po dílcích na jednotlivé dny by na jednu stranu znamenalo přímé zobrazování, ovšem za cenu

vysokého počtu dat v databázi a náročného načítání. Z výše uvedených důvodů byl nakonec zvolen model, kdy se ovlivnění plánů počítá pouze při uložení do databáze. Zároveň se však shlukují do intervalů, které mají shodné parametry. Za účelem možnosti vytvořit plán, který již nemůže být následně ovlivněn nově přidanými, byl do tabulky zahrnut sloupec typu boolean (bit).



Obrázek 2: Tabulka Plánů před a po změně

3.2 Aplikační vrstva

Na této vrstvě se změny týkaly především implementace nových algoritmů pro plánování. Původní byly nevyhovující z několika důvodů, především však kvůli špatnému počítání překrývajících se plánů. Hodiny na jednotlivé dny se často dostávaly do záporných hodnot, výpočty trvaly příliš dlouho a prováděly se při každém překreslení časové osy. Proto byla celá tato funkcionalita nahrazena novou. Implementace řešení problematiky plánování na aplikační vrstvě zabrala značnou část z celkové práce na systému. Použité algoritmy byly nejprve důkladně promyšlené, následně implementované a optimalizované s ohledem na časovou i paměťovou náročnost. Soustředila jsem se především na minimální počet přístupů do databáze. Proto pokud je to možné jsou všechna potřebná data načítána vždy na začátku prováděné operace a ukládána najednou při jejím ukončení. Všechny změny provedené nad databází jsou potvrzeny až po dokončení všech částí algoritmu, aby byla zachována konzistentnost databáze.

3.2.1 Řešení souběhu při plánování

V případě přidávání, editace, či mazání plánů je nutné zajistit, aby data, se kterými se pracuje, nebyla v průběhu algoritmu změněna. Může k tomu docházet převážně proto, že administrátory firmy může být více uživatelů zároveň. Ti mohou ve stejný čas plánovat práci zaměstnancům, měnit nastavení pracovní doby, či editovat projekty. Mohla by například nastat situace, kdy je přidán plán s určitým počtem hodin a před uložením jiný uživatel změní nastavení pracovní doby na daný den. To by mohlo znamenat práci přesahující pracovní dobu, u projektu by byly špatně spočteny naplánované hodiny apod. Bylo tedy nutné takovému chování zabránit.

```
public JsonResult AddTimelineItem(PlanningViewModel model) {
    try {
        TransactionOptions TransOpt = new TransactionOptions();
        TransOpt.IsolationLevel = IsolationLevel.RepeatableRead;
        using (TransactionScope scope = new TransactionScope(
            TransactionScopeOption.Required, TransOpt))
         {
          _timelineItemService.AddPlan(timelineItem, compId,0);
          scope.Complete();
          result.ErrorMessage = "Saved";
          result.IsSuccess = true;
          return Json(result);
         }
        }
     catch (UserShowableException e)
        {
         result.ErrorMessage = e.Message;
         result.IsSuccess = false;
         return Json(result);
        }
     catch (EntityException ex)
        {
         var e = ex.GetBaseException() as SqlException;
         if (e.Number == 1205)
            {
              result.ErrorMessage = "Someone changed data. Your changes wasn't
                  saved.";
              result.IsSuccess = false;
              return Json(result);
            }
         else throw ex;
        }
  }
```

Výpis 2: Ukázka použití TransactionScope pro nastavení stupně izolace

Výchozí stupeň izolace pro SQL Server je Read Committed, což pro tuto potřebu není dostačující. Potřebný je minimálně stupeň Repeatable Read [1]. Pro nastavení pouze u transakcí, které jej opravdu potřebují, jsem využila třídy TransactionScope. Při využití tohoto stupně izolace

může docházet k uváznutí (deadlocku). V tomto případě SQL Server ukončí jednu z transakcí a vyvolá výjimku typu SqlException s kódem 1205 [2]. EF ji zabalí do dalších výjimek s poslední typu EntityException. Tu zachytávám a získávám z ní prvotní. Následně ověřím kód definovaný pro deadlock u SQL Serveru. Pokud je výjimka jiného typu opětovně se vyvolá. Při uváznutí nemá smysl se pokoušet provádět operaci znova, protože data, se kterými uživatel pracoval, byla změněna. Funkce tedy končí a informace je zobrazena uživateli. Při použití jiného databázového systému nebude výjimka zachycena, protože použitý kód u deadlocku je takto definován pouze pro SQL Server. Chyba bude zapsána do logu a uživateli zobrazena informace o neuložení dat.

3.2.2 Přidání nového plánu

Funkce nejprve načte z databáze potřebná data, kterými jsou kolidující plány a nastavení časové osy v intervalu přidávaného plánu. Pokud zaměstnanec nemá nastavení pracovní doby pro celý interval plánu, není možné jej uložit. Tuto informaci bylo potřeba zobrazit uživateli. Pro tento, a jemu podobné případy, byla implementovaná třída UserShowableException odvozená z třídy Exception. V případě jejího vyvolání je zachytávána prvotními funkcemi algoritmu na Prezentační vrstvě, jak je ukázáno ve výpisu 2, a důvod jejího vyvolání je zobrazen uživateli. Tímto jsem dosáhla možnosti informovat uživatele o problému, který zabránil uložení plánu v kterékoli fázi algoritmu bez ztráty funkcionality zachytávání neočekávaných výjimek systémem.

Pokud má zaměstnanec všechna potřebná nastavení, zavolá se funkce, která ověří, zda požadované hodiny v jednotlivých dnech jsou volné. Ta má jako návratovou hodnotu typ DateTime a vrací první den, ve kterém se nový plán nevejde na časovou osu, případně následující den po konečném datu plánu.

```
private DateTime FindIfPlanFits(int userId, int compId, TimelineItem newPlan,
    List<TimelineItem> collidingItems, List<UserCompanyTimelineSettings>
    userSettings)
      {
            var actDate = newPlan.StartDate;
            for (; actDate <= newPlan.EndDate; actDate = actDate.AddDays(1))
            {
                var setting = userSettings.Single(s => s.Date == actDate);
                if (!setting.IsHoliday
                && (_companyTimelineService.GetUserFreeDayHours(setting, items,
                    userId, compId) < newPlan.DayHours)) return actDate;
            }
            return actDate;
        }
```

Výpis 3: Funkce ověřující dostupnost volných hodin pro nový plán

Pokud tato funkce vrátí následující den po konečném datu plánu, znamená to, že se vejde na časovou osu bez úprav. V tomto případě proběhne pouze slučování a uložení do databáze. V opačném případě se plánu změní konečné datum na poslední nekolidující den. Následně je vytvořen nový se stejnými parametry a počáteční datum je nastaveno na první kolidující den. Poté je volána obdobná funkce jakou ukazuje výpis 3, která ale naopak vrací první den, ve kterém je pro plán dostatečný počet hodin. Následně jsou všechny dny, ve kterých plán koliduje, vyřešeny samostatně, jak je popsáno v podkapitole 3.2.3. Tímto jsem zajistila, že časově i paměťově nejnáročnější část, tedy rozpočítávání kolidujících dní a jejich následné shlukování, je prováděna jen ve dnech, ve kterých je to nutné. Pokud takto nebyl vyřešen celý interval vytvoří se nový plán s počátečním datem, které odpovídá prvnímu nekolidujícímu dni a celý algoritmus mimo načítání dat se opakuje.

Po úspěšném vyřešení celého intervalu je provedeno slučování těch, které představují práci na stejném projektu ve stejný den a nejsou označeny jako needitovatelné. Následně jsou plány slučovány do intervalů o odpovídajících parametrech. Pokud jsou nalezeny dva plány se stejným počtem hodin, projektem, zaměstnancem a hodnotou určující možnost editace, kdy jeden plán končí den před začátkem druhého, jsou tyto sloučeny v interval. Po úspěšném vyřešení jsou všechny nové části uloženy do databáze a plány označené ke smazání (byly nahrazeny novými s jinými parametry) jsou odstraněny.

Následující tabulky ukazují příklad plánů jednoho zaměstnance uložených v databázi. Pro zjednodušení nejsou v tabulce uvedeny sloupce id plánu a zaměstnanec.

Tabulka 1: Příklad obsahu tabulky plánů

Projekt	**Začátek**	**Konec**	**Počet hodin**	**Editovatelný**
1	01.04.2019	05.04.2019	3	Ano
2	02.04.2019	03.04.2019	2,5	Ano
1	01.04.2019	04.04.2019	1	Ne
2	04.04.2019	09.04.2019	1,5	Ano

Po úspěšném zpracování požadavku na přidání nového plánu na projektu číslo dvě, který začíná 01.04.2019, končí 03.04.2019, je editovatelný a požadovaný počet hodin je 5, bude obsah tabulky následující.

Tabulka 2: Obsah tabulky plánů po přidání nového

Projekt	**Začátek**	**Konec**	**Počet hodin**	**Editovatelný**
1	01.04.2019	01.04.2019	3	Ano
2	01.04.2019	01.04.2019	4	Ano
1	01.04.2019	04.04.2019	1	Ne
1	02.04.2019	03.04.2019	2,33	Ano
2	02.04.2019	03.04.2019	4,66	Ano
1	04.04.2019	05.04.2019	3	Ano
2	04.04.2019	09.04.2019	1,5	Ano

Pokud by nyní uživatel editoval plán, který je uveden na třetím řádku v tabulce 2, a nově by ho označil jako editovatelný, přičemž ostatní parametry by zůstaly nezměněny, byl by požadavek zpracován následovně. Nejprve by byl porovnán s původním, jak je uvedeno v podkapitole 3.2.4. Na základě toho by byl starý plán z databáze smazán a editovaný přidán jako nový. Díky nezměněným hodinám by bylo zjištěno, že plán se vejde na časovou osu beze změn. Proběhlo by tedy slučování shodných projektů v jednotlivých dnech a následně také intervalů. V databázi by poté byly uloženy následující plány.

Tabulka 3: Obsah tabulky plánů po editaci jednoho z nich

Projekt	**Začátek**	**Konec**	**Počet hodin**	**Editovatelný**
1	01.04.2019	01.04.2019	4	Ano
2	01.04.2019	01.04.2019	4	Ano
1	02.04.2019	03.04.2019	3,33	Ano
2	02.04.2019	03.04.2019	4,66	Ano
1	04.04.2019	04.04.2019	4	Ano
2	04.04.2019	09.04.2019	1,5	Ano
1	05.04.2019	05.04.2019	3	Ano

3.2.3 Řešení kolidujících plánu

Tato funkce má za vstupní parametry přidávaný plán (již upravený na interval ve kterém koliduje), kolidující plány v tomto intervalu a nastavení časové osy zaměstnance. Vrací soubory plánů k přidání a ke smazání z databáze. V první fázi všechny kolidující plány rozdělí v intervalu nového plánu. Nově vytvořené v intervalech mimo přidávaný plán jsou přidány do seznamu pro uložení do databáze. Následně je každý den intervalu vyřešen samostatně.

Kolekce plánů (kolidujících a přidávaného) je procházena celkem třikrát. Nejprve se pro každý item vytvoří nový s počátečním i koncovým datem odpovídajícím aktuálnímu dni cyklu. Pokud tento plán má hodnotu indikující, zda smí být automaticky měněn nastavenou na false, pak jsou od pracovních hodin daného dne odečteny jemu odpovídající hodiny. Jestliže se jedná o přidávaný item a má počet hodin vyšší než jsou zbývající volné hodiny nastaví se jejich počet na tuto hodnotu a cyklus pokračuje dalším dnem. V opačném případě funkce spočte hodiny pro jednotlivé plány vydělením počtu hodin, které nejsou zabrány needitovatelnými, počtem plánů, které se můžou upravovat a přechází k dalšímu průchodu kolekcí. V této fázi najde itemy, které je možné editovat, a které mají počet hodin nižší než je hodnota vypočtena pro jeden plán. Všechny přesahující hodiny jsou sečteny a funkce přechází k poslednímu průchodu, ve kterém přiděluje hodiny zbylým plánům. Ty jsou vypočteny součtem dříve zjištěných hodin pro jeden plán a přičtením počtu přesahujících hodin opět vyděleným počtem zbývajících plánů.

Všechny plány mají hodiny zaokrouhlené na dvě desetinná místa. Ty, které mají rozsah méně než půl hodiny jsou smazány.

3.2.4 Editace existujícího plánu

Nejprve se načte původní plán z databáze. Ten je následně porovnán s novým. Pouhé uložení změn do databáze proběhne v případě, že jsou změny v následujícím rozsahu:

1. Byl změněn projekt
2. Nové počáteční datum je větší než původní
3. Nové konečné datum je menší než původní
4. Nový počet hodin je nižší
5. Plán byl nově označen jako needitovatelný

Navíc je upraven počet naplánovaných hodin příslušného projektu. Pouhé uložení platí při jakékoli kombinaci jedné nebo více z výše uvedených změn. Pokud kterákoli změna není v souladu s výše uvedeným (například byl změněn zaměstnanec, nebo je počet hodin větší) je původní plán smazán a editovaný přidán jako nový algoritmy popsanými výše.

```
public bool NeedNewResolve(TimelineItem newPlan, TimelineItem oldPlan)
        {
            if (newPlan.User.Id != oldPlan.User.Id ||
                newPlan.DayHours > oldPlan.DayHours ||
               (newPlan.EnableAutoChange && !oldPlan.EnableAutoChange) ||
                newPlan.StartDate < oldPlan.StartDate ||
                newPlan.EndDate > oldPlan.EndDate) return true;
            else return false;
        }
```

Výpis 4: Funkce ověřující, zda je nutno nový plán znovu rozpočítat

3.2.5 Změna nastavení pracovní doby zaměstnance

V aplikaci je možné zaměstnanci měnit nastavení pracovní doby. Den se dá označit za nepracovní, nebo mu změnit počet hodin. V případě, že na tento den již existují pro daného uživatele plány přesahující novou požadovanou pracovní dobu, nemůže být takovému požadavku vyhověno. První možností bylo plány na změněný den znovu rozpočítat s ohledem na požadovanou pracovní dobu a provést všechny změny s tím související. Zadavatel systému však požadoval pouze zamezení možnosti měnit pracovní dobu dní, u kterých existují přesahující plány. Při editaci je na server posílán model obsahující nastavení pro všechny dny daného roku. Z databáze se načtou ty původní a jednotlivé dny jsou porovnávány. V případě sníženého počtu hodin, nebo změny na nepracovní den se zjišťují již naplánované hodiny. V případě přesahu nejsou uloženy

žádné změny a uživatel informován o nemožnosti změnit tento den. Opět je zde využito třídy UserShowableException ke sdělení důvodu ukončení.

```
if (newSetting.WorkingHours < currentSetting.WorkingHours
     || (newSetting.IsHoliday && !currentSetting.IsHoliday))
    {
       var plannedHours = _timelineItemService.GetUserPlannedDayHours(
           currentSetting.Date, userId, companyId);
       if ((newSetting.IsHoliday && (plannedHours > 0.01))
           ||(plannedHours > newSetting.WorkingHours))
          throw new UserShowableException("User already have plans on day " +
             currentSetting.Date.ToString("dddd, dd MMMM yyyy") + ". Cannot
             change settings.");
     }
```

Výpis 5: Část kódu ověřující možnost změny pracovní doby daného dne.

3.2.6 Odebrání zaměstnance z firmy

Při odebrání zaměstnance z firmy jsou smazána všechna data z tohoto vztahu vyplývající. Tedy nastavení pracovní doby, plány, zařazení do center, k projektům apod. Při mazání plánů je současně nutno měnit naplánované hodiny příslušných projektů. Aby tato informace u projektu byla logicky správná, jsou ubrány pouze hodiny plánů, které reprezentují budoucí práci. Uplynulé dny včetně aktuálního jsou považovány za odpracované, tudíž práce na projektech proběhla a je uchována v naplánovaných hodinách projektu. Plány jsou ovšem mazány všechny, protože je není možné uchovávat bez vazby na zaměstnance.

```
public void RemoveSheduledHoursFromNow(List<TimelineItem> items, int compId)
       {
        var date = DateTime.Now;
           var futureItems = items.Where(i => i.EndDate >= date);
           foreach (var item in futureItems)
           {
               _projectService.ChangeSheduledHours(item.Project.Id, -
                  GetPlanWorkingHoursFromNow(item, compId));
           }
       }
```

Výpis 6: Funkce, která odečte projektům budoucí naplánované hodiny zaměstnance.

3.2.7 Formátování plánů pro vykreslení

Jsou implementovány tři třídy pro formátování plánů jednotlivých časových os.

1. Pro časovou osu firmy

   Pro firmu jsou data formátována do seznamu jednotlivých zaměstnanců a k nim příslušných nastavení pracovní doby a dílků plánů. Z každého plánu se vytvoří dílky pro každý den z jeho intervalu. Ty nesou informaci o dni v měsíci ke kterému přísluší, jméno projektu, barvu projektu, počet hodin a Id plánu. Den se může ukládat jako číslo dne v měsíci, protože časové osy na prezentační vrstvě zobrazují maximálně jeden měsíc najednou a tudíž je tato informace jednoznačná. Dílky jsou seřazeny podle barvy, aby po postupném vykreslení byla osa přehlednější. Nastavení pracovní doby nesou taktéž číslo dne v měsíci a příslušný počet hodin, nebo informaci, že den není pracovní či nemá nastavení. V případě, že zaměstnanec nemá nastavení pro celý formátovaný interval, je tato skutečnost uchována v doplňující zprávě ve struktuře a nastavení pracovní doby ani dílky plánů nejsou přenášeny.

2. Pro časovou osu centra

   Pro plánování centra jsou data formátována do seznamu projektů a příslušných dílků. Dílky jsou z plánů vytvářeny pouze v pracovních dnech příslušného zaměstnance a nesou obdobné informace jako při plánování firmy. Místo barvy a názvu projektu však obsahují jméno a barvu zaměstnance.

3. Pro časovou osu uživatele (zaměstnance)

   Pro uživatele jsou data formátována podobně jako pro firmu, ale pouze pro jednoho daného uživatele. Nejsou tedy přenášena ve formě seznamu zaměstnanců, ale jako struktura obsahující seznam dílků a nastavení pracovní doby aktuálního uživatele. Časová osa uživatele je zobrazována pouze jako týdenní, místo dne v měsíci tedy data obsahují den v týdnu.

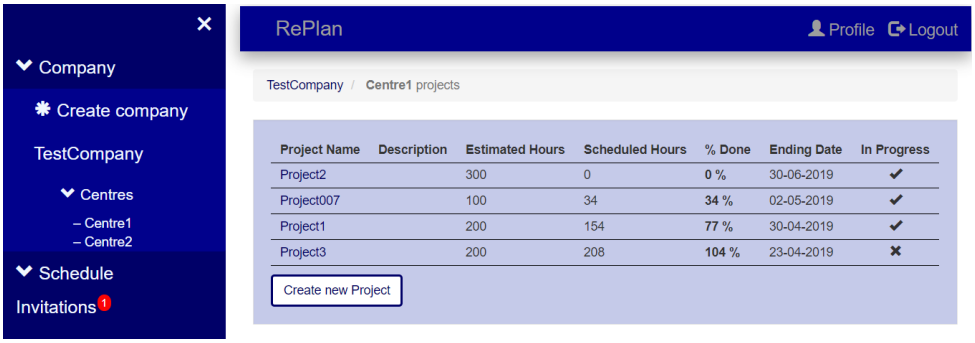
3.3 Prezentační vrstva

V první fázi jsem se zabývala úpravou grafického rozhraní a návaznosti aplikace. Hlavní a časově nejnáročnější na této vrstvě byla implementace časových os a plánování. K dispozici jsou časové osy pro aktuálního uživatele, firmu a jednotlivá centra. Původní aplikace využívala vizualizační knihovnu Vis, konkrétně její komponentu Timeline. Nejprve jsem pracovala s možností, že bude využívána i nadále. Přepracovala jsem JS funkce, které ji obsluhovaly na novou implementaci plánování. Původně byla načítána data pro celý rok najednou, což bylo ve většině případů nepotřebné, trvalo to příliš dlouho a přenášelo se velké množství dat. Pro zrychlení jsem nově načítala pouze data, která byla potřeba pro aktuální zobrazení. To ovšem způsobovalo načítání nových dat při každém posunu osy, byť o jeden den. Proto jsem implementovala JS funkce, které načítaly data vždy navíc o sedm dní před i za zobrazovaným intervalem. Až v případě překročení

tohoto intervalu byla načtena nová. To velmi zrychlilo aplikaci a snížilo množství přenášených dat. Zobrazování časové osy pomocí knihovny Vis.js však nebylo vyhovující. Jednotlivé dílky se nedaly upravovat, nezobrazovaly se v řádcích jak měly a ani po důkladném prostudování dokumentace se nepovedlo docílit potřebných výsledků. Po konzultaci se zadavatelem požadavků na systém bylo dohodnuto, že ve výsledné aplikaci nebude použita a implementuji vlastní JS funkce pro vykreslování. Jedním z důvodů byla také nejistota budoucí podpory této knihovny a fakt, že bylo využíváno pouze minimum funkcí, které nabízela. Vlastní implementaci popíšu níže.

3.3.1 Úprava grafické stránky UI

V aplikaci byla využívána knihovna Bootstrap, ale pouze pro některé její předdefinované styly. Při úpravě grafického rozhraní jsem využila tuto knihovnu ve větší míře. V první fázi jsem UI upravila tak, aby bylo responzivní. Primárně ne pro podporu zobrazení na mobilních telefonech, ale pro konzistentní vzhled při různých velikostech okna prohlížeče. Díky využití Bootstrap se jednalo pouze o změnu některých použitých tříd na jednotlivých stránkách a úpravy tabulek. S touto knihovnou jsem již měla zkušenosti z vývoje webových stránek, takže to pro mě nebylo nijak obtížné. Dále bylo lehce upravené barevné schéma celé aplikace a zvýšen kontrast jednotlivých prvků, jako například tlačítek. Pro dosažení intuitivního rozhraní jsem využila na některých místech ikon (Glyphicons) obsažených v knihovně.



Obrázek 3: Ukázka rozložení UI a seznamu projektů

3.3.2 Návaznost aplikace

V aplikaci byla implementovaná takzvaně drobečková navigace. Tu jsem upravila a dokončila tak, aby se z ní dalo vždy vrátit o úroveň výš, což v původní verzi na mnoha místech nebylo možné. Dalším krokem pro zjednodušení používání aplikace hlavně novým uživatelům bylo přidání tlačítka na řešení aktuálního problému. Například při pokusu zobrazit časovou osu, když uživatel nepatří do žádné firmy, je zobrazena informace, že musí nejdříve založit novou firmu a tlačítko pro přechod na stránku, kde si ji může vytvořit. Takto byly vyřešeny všechny tyto situace. Pokud tedy přijde nový uživatel, který nemá s aplikací žádné zkušenosti, tak ta ho

postupně nasměruje na jednotlivé kroky k úspěšnému používání všech funkcionalit. Na mnoha místech jsem upřesňující informace schovala pod značku otazníku, aby zbytečný text nerušil zkušené uživatele, ale zároveň si nový mohl vše jednoduše ujasnit. Celou aplikaci jsem následně nechala odzkoušet lidmi, kteří ji nikdy nepoužívali, abych ověřila, že bylo dosaženo požadovaných výsledků.

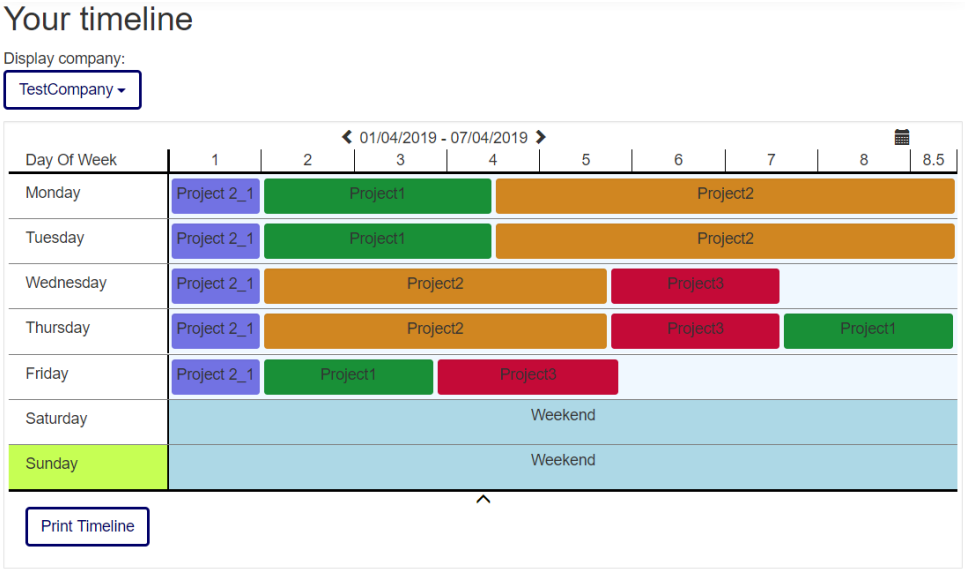
3.3.3 Boční navigace

Pro rychlé přechody mezi částmi aplikace byla implementována skrývatelná boční navigace, ze které se uživatel jednoduše dostane ke správě firem, center či k plánování. Využila jsem tříd z knihovny Bootstrap a implementovala vlastní JS funkce pro její ovládání. Při otevření posune obsah stránky o svoji šířku, takže je celý k dispozici. Při zobrazení na malých displayích se otevírá přes celou šířku. K ovládání slouží dvě tlačítka. Pro dotykové zařízení jsou navíc implementované JS funkce, které umožňují otevírání a zavírání přetažením po displayi. Na velkých obrazovkách je automaticky otevřená, na středních a malých zavřená. Toho bylo docíleno pomocí CSS. U sekce pozvánek jsem vytvořila element indikující, že má uživatel nějakou nevyřízenou pozvánku. Je zobrazen jejich počet v červeném kruhu jak je ukázáno na obrázku 3. Pokud žádnou nemá, není zobrazen vůbec.

3.3.4 Časová osa uživatele

Časová osa uživatele je zobrazena na profilové stránce. Pro přehlednou strukturu kódu a znovupoužitelnost jsem využila částečné stránky MVC (Razor). Komponentu časové osy jsem vytvořila v novém HTML souboru a ten poté přidala do stránky s profilem. Osa pro uživatele zobrazuje vždy jeden týden a je možné zobrazovat ji pro kteroukoli firmu, do které uživatel patří. Je formátována jako tabulka, kdy řádky představují jednotlivé dny a pracovní doba je vyjádřena v šířce sloupce. Počet hodin dílku plánu je vyjádřen jeho šířkou. U tohoto zobrazení není možno plány nijak editovat.

Základ časové osy, který se nemění, tedy jednotlivé řádky pro dny, levý sloupec obsahující jejich názvy a prázdné elementy pro hlavičku, včetně ovládacích prvků jsou zapsány v HTML souboru. Výchozí týden je vždy ten aktuální. Při úvodním vykreslení a změně zobrazovaného týdne je zasílán AJAX požadavek na server s daty požadovaného intervalu, Id uživatele a zobrazované firmy. Ze serveru jsou data pro časovou osu zaslána opět ve formátu JSON pomocí AJAX. Díky tomu není při změně načítána stránka znovu, ale je pouze překreslena osa [3]. Data jsou zpracována implementovanými JS funkcemi. Ty upravují a vytváří zobrazení pomocí DOM elementů, což je API umožňující přístup či modifikaci obsahu dokumentu [4]. Využívám také jQuery pro získávání elementů ze struktury HTML. Nejprve se z nastavení pracovní doby pro daný týden vybere maximum a vytvoří se hlavička indikující hodiny. Krok je po jedné hodině, případně je na konci zbytek přesahující celé hodiny. Kontejner dílků jednotlivých dnů je poté upraven podle příslušné pracovní doby. Tu představuje mírně ztmavené pozadí, pokud den není



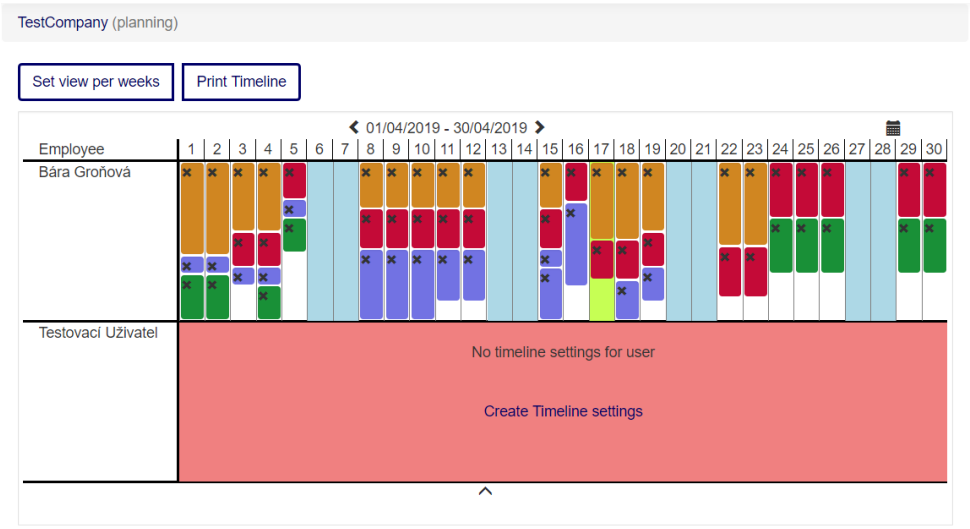
Obrázek 4: Časová osa uživatele

pracovní je mu přidána příslušná třída a vepsán text informující o typu nepracovního dne. Pokud den nemá nastavení časové doby je tam tato informace vypsána a kontejner je červený. Opakující se styly jsou zapsány v CSS souboru a elementům jsou přidávány příslušné třídy. Každý kontejner pro daný den si nese informaci o pracovní době. Následně jsou zpracovány postupně všechny dílky plánů. Pro každý je vytvořen nový element v příslušném dni. Jeho šířka se počítá z pracovní doby daného kontejneru a počtu hodin dílku. Je nastavena jeho barva a také barva rámečku, který je využíván při tisku. Nakonec se přidá text se jménem projektu. Pokud zobrazovaný týden obsahuje aktuální den, je pozadí jeho řádku zvýrazněno. Při překreslování osy jsou nejprve odstraněny všechny elementy v denních kontejnerech a následně vytvořeny nové na základě nových dat. V pravém horním rohu osy je ikona kalendáře, po kliknutí se zobrazí kalendář, ve kterém se dá vybrat zobrazovaný týden. K tomu využívám komponentu Datepicker z knihovny jquery-ui. Ta je využívána ve všech částech aplikace, kde se volí datum.

3.3.5 Časová osa firmy a centra

Časová osa firmy a centra je velice podobná. Rozdíl je v tom, že pro firmu jsou zobrazované plány na projektech pro zaměstnance a pro centrum naopak, kdy na jednotlivých projektech pracují zaměstnanci. Proto je většina funkcí, které pro zpracování a vykreslení dat využívají společná v jednom JS souboru. Taktéž je společná částečná stránka s časovou osou, ve které se pouze programově mění v záhlaví legendy osy zaměstnanec na projekt. U firmy jednotlivé řádky představují časové osy zaměstnance, u centra projektu. Sloupce pak představují jednotlivé dny. Zobrazení je možné měsíční, nebo týdenní. Počet hodin daného plánu je vyjádřen výškou dílku plánu. Toto platí pouze u plánování firmy. U centra tato informace není ze zobrazení patrná, dílky jsou všechny stejně velké nehledě na počet pracovních hodin a představují pouze, že v ten

den zaměstnanec na daném projektu pracuje. U těchto zobrazení je možná editace plánu. Každý denní dílek plánu má ikonu pro jeho smazání. Editace celého plánu (intervalu) je možná dvěma způsoby. Buď dvojklikem na dílek daného plánu, nebo jeho přetažením (drag and drop) do editačního formuláře, který je implementován pod časovou osou a bude popsán níže. V obou případech je pomocí AJAX načten daný plán z databáze podle jeho Id. Každé zobrazení časové osy má vlastní adresy pro načítání a odesílání dat na server, ale zpracovávají se obdobně.



Obrázek 5: Ukázka časové osy firmy

HTML soubor se základem časové osy obsahuje pouze element pro osu samotnou a předpřipravené záhlaví pro její ovládání. Celá osa se vykresluje dynamicky na základě dat získaných ze serveru. Nemůžou být předpřipravené jednotlivé dny ani řádky pro zaměstnance, respektive projekty, protože jejich počet záleží na aktuálním režimu zobrazení a počtu zaměstnanců, respektive projektů. Na základě dat jsou tedy nejprve vytvořeny jednotlivé řádky, v nich kontejnery pro dny a inicializovaná hlavička. Pokud je aktuální zobrazení týdenní je dní sedm, pokud měsíční tak tolik, kolik jich je v daném měsíci. Následně jsou procházeny všechny dílky a jsou jim vytvářeny elementy v příslušných kontejnerech. Jejich výška je daná procentuálně na základě pracovních hodin v poměru k pracovní době dne. U plánování centra je výška konstantní. Kromě barvy pozadí je nastavena barva rámečku kvůli tisku. Každému dílku jsou také přiřazeny metody pro dvojklik a přetažení. Je vytvořena editační ikona uvnitř jeho elementu, která umožňuje dílek smazat. Před smazáním je vyžadováno potvrzení, k tomuto jsem využila funkce Confirm. Také je mu přidána hodnota takzvaně title. Do té je uloženo jméno projektu, respektive zaměstnance a počet hodin. Toto zajišťuje, že po najetí myší na daný dílek se tato informace zobrazí. Kontejnery nepracovních dní u firemního plánování mají ztmavené pozadí. Pokud uživatel nemá nastavení pracovní doby, je pozadí červené a informace v něm vypsána spolu s odkazem na vytvoření. Při překreslování časové osy jsou všechny vytvořené elementy smazány a nahrazeny na základě nových dat.

3.3.6 Formulář pro vytváření a editaci plánů

Přidávání a editace plánů je možná ve formuláři, který je umístěn pod časovou osou firmy a centra. Je uložen v částečné stránce a načítán do jednotlivých zobrazení. U výběru projektu bylo třeba, aby v nabídce nebyly zobrazeny již ukončené projekty, ale i tyto musely být dostupné kvůli možnosti editace plánu tohoto projektu. Bylo tedy potřeba pouze zamezit jejich výběru, ale při načtení plánu s tímto projektem ze serveru jej zobrazit. První možností bylo těmto projektům v seznamu přiřadit hodnotu „hidden“. Toto se ovšem ukázalo jako nefunkční při použití IE. Hodnota „readonly“ nemohla být použita, protože takto označená položka se při odeslání formuláře neukládala. Vyřešila jsem to tak, že ukončené projekty v seznamu nejsou, ale při načtení plánu ze serveru JS funkce ověří, zda je projekt v seznamu. Pokud ne, tak jej přidá a zapamatuje si to. Po odeslání formuláře jej ze seznamu opět odebere.

Při dvojkliku na plán z časové osy, nebo jeho přetáhnutí do formuláře, je načten plán ze serveru a inicializován formulář. Aby bylo indikováno načtení hodnot do formuláře je po krátkou dobu změněna barva jeho pozadí. Vyplněné hodnoty jsou validované na straně klienta pomocí komponenty jquery Validation. Po úspěšném vyplnění formuláře jsou data odeslány za pomocí AJAX ve formátu JSON na server. Pro přidání i editaci je použita stejná funkce. Při editaci model obsahuje také Id editovaného plánu, podle kterého obsluhující funkce na serveru zavolá metodu pro editaci místo přidávání. Po zpracování dat server zasílá zpět zprávu o výsledku. Ta obsahuje informativní text a hodnotu indikující, zda bylo uložení úspěšné. K jejich zobrazení využívám element, který je jinak skrytý a zobrazuje se pouze pro tento účel. Je umístěn v horní části stránky a je částečně průhledný. Podle toho, zda byla operace úspěšná, je buď červený, nebo zelený a obsahuje text zprávy. Je zobrazen pouze po krátký čas v závislosti na délce textu zprávy tak, aby uživatel stihl zprávu bez problému přečíst, ale aby nepřekrýval část obsahu po zbytečně dlouhou dobu.

3.4 Přidaná funkcionalita

Do výsledné aplikace byly také přidány nové funkce pro zjednodušení práce a rozšíření možností. Ty nejzajímavější a jejich řešení shrnu v této podkapitole.

3.4.1 Pozývaní neregistrovaných uživatelů přes email

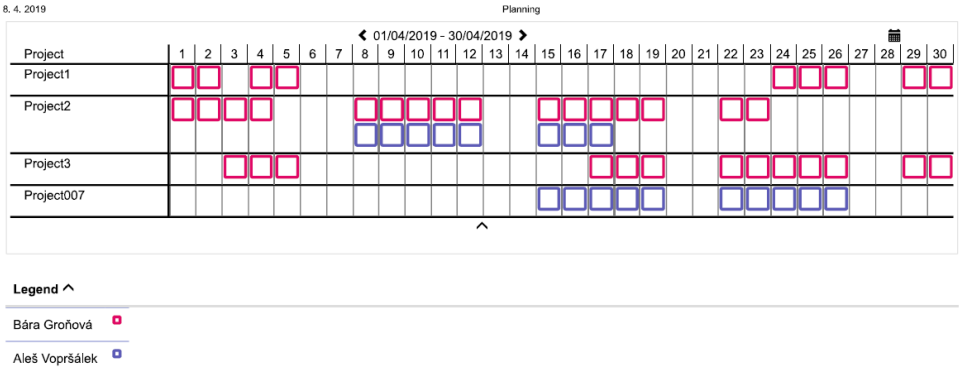
V původní aplikaci bylo do firmy možno pozvat registrované uživatele, nebo registrovat nového se všemi informacemi včetně hesla. Novým požadavkem bylo na základě zadané emailové adresy, která není v aplikaci registrovaná, odeslat email s odkazem k registraci, po které bude uživatel pozvaný do dané firmy.

Po zadání neregistrovaného emailu je vytvořena URL adresa, která obsahuje daný email a ID firmy, která uživatele pozvala. Pomocí něj se uživatel z obdrženého emailu dostane na stránku k registraci s předvyplněným emailem, který nelze měnit. Pokud email obsažený v adrese v databázi již existuje je vrácena chyba (404 Nenalezeno). Tím je docíleno pouze jednorázově

platného odkazu. Po úspěšném vyplnění formuláře jsou data uložena do databáze a vytvořen záznam v tabulce pozvánek s ID firmy z url adresy. Účet je automaticky ověřený. Nyní se již uživatel může přihlásit a v sekci pozvánek najde danou firmu.

3.4.2 Možnost tisku časových os

Jedním z požadavků na novou aplikaci byla možnost tisku časových os s legendou. Toho bylo docíleno pomocí CSS. V něm je nastavena u tisku pro celý dokument vlastnost, která skryje jeho obsah. Časovým osám (a sekcím s legendou) byla v HTML přidána třída „printing-area“, která má v CSS vlastnost pro tisk nastavenou na viditelnou. Dále je nastavena pozice této oblasti na levý horní roh. Problémem bylo, že pro přehlednost časové osy jsou nutné barvy pozadí jednotlivých prvků. Výchozím nastavením prohlížečů však je toto do tisku nezahrnovat. Zároveň by bylo zbytečné tisknout tak velké plochy barev. Proto byl dílkům časové osy a indikátorům legendy přidán rámeček, který má stejnou barvu jako pozadí a šířku 3 pixely. Ikony pro mazání itemů jsou pro tisk skryté. Lze jej vyvolat klasickým vytištěním stránky, nebo tlačítkem umístěným nad časovou osou, které pomocí JS funkce vytiskne okno. Před tiskem je vždy nejdřív otevřena legenda.



Obrázek 6: Příklad tisku časové osy centra

4 Závěr

Výsledný systém byl nasazen na firemní server a otestován. Finální verze odpovídá požadavkům zadavatele a byla schválena. Při práci na aplikaci, kterou v minulosti napsal jiný student a neexistovala k ní dokumentace, jsem si uvědomila nejen důležitost používání jednoznačného a logického pojmenovávání, ale také užitečnost komentářů k jednotlivým funkcím. Snažila jsem se zachovat členění aplikace i její názvosloví, aby případný budoucí vývoj byl co nejjednodušší.

4.1 Uplatněné znalosti získané v průběhu studia

Při úpravách datové vrstvy byly nezbytné znalosti z předmětů UDBS a DAIS, především problematika objektově relačních mapperů a postupů při návrhu databáze. Serverová část aplikace je implementovaná na frameworku ASP.NET MVC v jazyce C# , znalosti platformy .NET jsem čerpala z předmětu PJ2. Se vzorem MVC jsem se poprvé setkala v předmětu SPJA na frameworku Django a následně v předmětu VIS při tvorbě jednoduché webové aplikace. Při úpravách grafického rozhraní jsem využila vědomostí z předmětu URO.

4.2 Chybějící znalosti

Již v začátku mé práce byly znát nedostatečné znalosti hlavně ohledně EF a fungování Razor stránek využívaných v ASP.NET MVC. To mi značně ztěžovalo pochopení fungování původní aplikace, hledání problémů a jejich řešení. Potřebné znalosti jsem získala studováním ukázkových řešení a oficiální dokumentace [5] [6]. S JS, Bootstrapem a obecně tvorbou webových stránek jsem měla základní zkušenosti z vlastních projektů mimo studium, tudíž jsem je pouze prohloubila. S knihovnou Vis.js, kterou využívala původní aplikace a já s ní také určitou dobu pracovala, jsem se setkala poprvé. Existovalo k ní velmi málo článků a ukázkových řešení, tudíž hlavním zdrojem informací byla oficiální dokumentace [7].

4.3 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

V průběhu odborné praxe jsem musela brát v potaz přání zadavatele a zároveň sama zvažovat možné potřeby budoucích uživatelů. Předběžná představa o výsledném systému byla nastíněna před začátkem vývoje a upřesňována postupně s každou hotovou částí. Tím se mi nehromadila práce a povedlo se mi dokončit systém ke spokojenosti zadavatele. Konzultant, který byl současně zadavatelem práce, byl také otevřený mým nápadům a implementační část nechal v mé režii. Zároveň však byl on i ostatní kolegové ochotni pomoci a poradit s jakýmkoli problémem, nebo nejasností.

Celkově hodnotím praxi velmi kladně a považuji ji za velký přínos do budoucna. Měla jsem možnost pracovat na reálném systému, který se nyní využívá pro interní potřeby firmy. Zároveň jsem mohla nahlédnout do fungování společnosti, řízení projektů a vývoje softwaru.

Literatura

[1] HENDERSON, Ken. *Mistrovství v Transact-SQL.* Přeložil Karel VORÁČEK. Praha: Computer Press, 2000. Databáze. Profi. ISBN 80-7226-393-5.

[2] KEHAYIAS, Jonathan. *Handling Deadlocks in SQL Server* [online]. 2012-03-10, [cit. 2019-04-17] Dostupné z: <https://www.red-gate.com/simple-talk/sql/database-administration/handling-deadlocks-in-sql-server/>

[3] RESIG, John. *JavaScript a Ajax: moderní programování webových aplikací.* Přeložil Ondřej BAŠE, přeložil Ondřej ŽIŽKA. Brno: Computer Press, 2007. ISBN 978-80-251-1824-5.

[4] FLANAGAN, David. *JavaScript: kompletní průvodce.* 2. aktualiz. vyd. Přeložil David KRÁSENSKÝ, přeložil Karel VORÁČEK. Praha: Computer Press, 2002. ISBN 80-7226-626-8.

[5] *Overview - EF6 | Microsoft Docs* [online]. 2016-10-23, Poslední revize 2019-10-14 [cit. 2019-04-17]. Dostupné z: <https://docs.microsoft.com/en-us/ef/ef6/>.

[6] *ASP.NET MVC Pattern | .NET* [online]. ©2019, [cit. 2019-04-17]. Dostupné z: <https://dotnet.microsoft.com/apps/aspnet/mvc>.

[7] *timeline - vis.js - A dynamic, browser based visualization library* [online]. Poslední revize 2018-07-14 [cit. 2019-04-17] Dostupné z: <http://visjs.org/docs/timeline/>.

Přílohy

I. Příloha v IS EDISON. Zdrojové kódy aplikace RePlan